REPORT

Reinforcement learning project

Reinforcement learning for lane change on a highway stretch

Supervisor:

DR. Nadir Farhi

Elaborated by:

Aissaoui Ilhem
Diab Bilal

Master 2 SIA
2021/2022

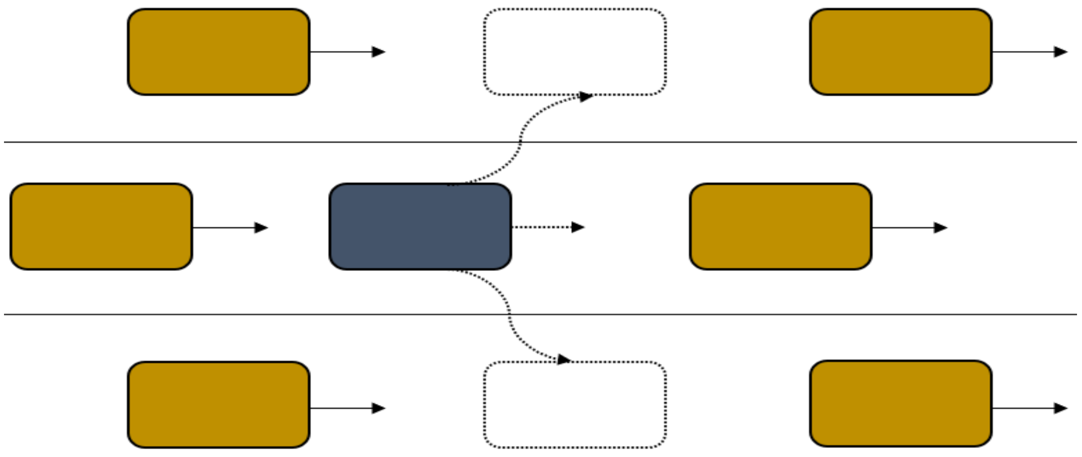
TABLE OF CONTENT

Introduction

In the field of autonomous robots and decision making, especially in autonomous cars the agent is required to learn new configurations of the environment, to predict an optimal decision at each instant while driving. This decision is in the form of an action taken by the vehicle in a certain moment considering the environment.

Through this project, the goal is to get a car to learn how to self-drive on a road applying Deep Reinforcement Q-learning methods, combining Reinforcement Learning and Deep learning. The simulation will be done on the SUMO traffic simulator and with the help of TraCi a python library.

We consider a stretch of highway with a given number of lanes and we have a certain number of cars following each other on a road where passing is done by lane change. We will focus on one car which we control. Using the Deep Q-Learning algorithm, the car will have to make a decision and change or not the lane, it can go to the left, to the right or remain in the same lane.



The project aims to develop a reinforcement learning application for an agent to drive safely in dense traffic conditions. In a dense traffic situation. To do this, SUMO was used to simulate the behavior of the target vehicle and a network of autonomous vehicles and to train a model of reinforcement learning algorithms, namely DQN.

The code is available here https://github.com/Ilhem23/change_lane_DQN

I. Working Environment

1. Sumo

Simulation of Urban Mobility is an open source, traffic simulation package designed to manage networks and model large roads, it provides a set of tools for scenario creation. and was created in order to simulate traffic scenarios

Sumo is the simulator, but to create the environment we use the visual network editor Netedit, it gives the possibility to create objects and structures including vehicles, pedestrians, infrastructures, intersections, approaches, lanes, connections, roads, highway, crossings, stops, sidewalks.

From Netedit, an XML file is generated, this file can be run from Sumo, or from python using an API called TraCi.

2. TraCi

Traci is an API used in python which allows the access to the XML file which contains the information about the network

By using it, all the data about the roads, the lanes and the cars can be recovered.we can also execute decisions on the model and train some elements such as the cars. In this kind of situation, the reinforcement learning is used.

3. OpenAi Gym

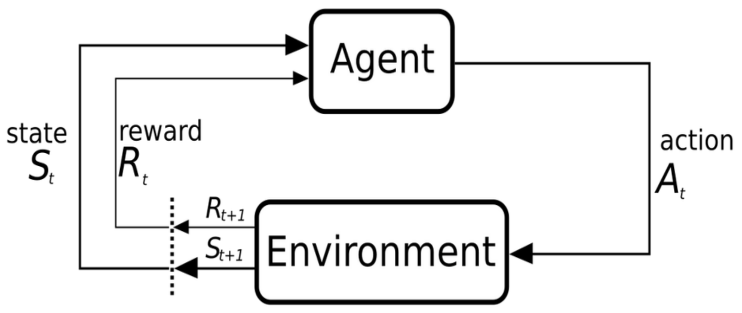
Gym is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games, simulate environments with many features, and writes the test logs.

II. Reinforcement learning

Reinforcement learning is a machine learning training method based on rewarding desired behaviors and punishing negative behaviors, it follows the logic of markov chains. the agent can perceive its environment, take actions and learn through some rules, errors and rewards, it seeks the maximum reward to achieve an optimal solution. With time, the agent learns to avoid the negative and seek the positive.

When the agent is in stat S, and using a q-table which groups all possible actions according to a given state, it will take the best action according to the maximum reward.

When the environment is complex or non-deterministic and the number of actions and states reach big numbers, evaluating every possible state-action pair can become impossible and the q-table could reach sky-high sizes, Deep Q learning is used.

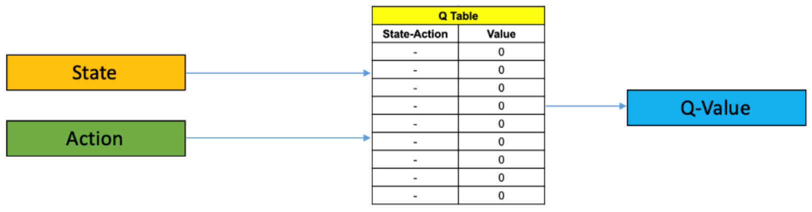


1. States, rewards, and actions

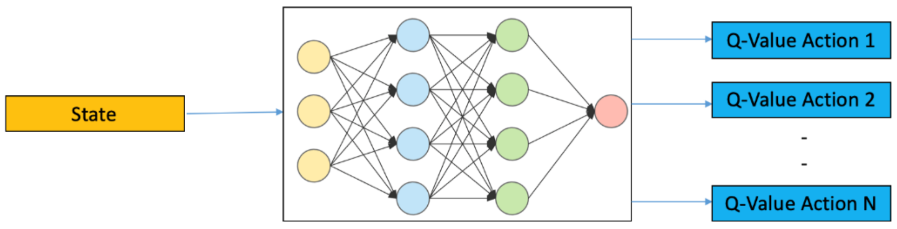
1. States: is the description of the environment at time t.
2. Reward: is value that is received by the agent at each transition from one state to another and thus by applying an action.
3. Action: it's a behavior of the agent which causes a transition from a state to another, Actions can be discrete (flipping a switch) or continuous (turning a knob).

III. DQN (Deep Q Learning)

This method combines the Q-learning algorithm with deep learning. Neural networks are used. In this case it is necessary to find an approximation function that can learn optimal policies without going through all the possible states of a system and by collecting data during training to finally find the best behaviour and the optimal policies.



Q Learning



Deep Q Learning

The output of this model is the generation of actions, and based on the feedback from the environment, the parameters of all the neural networks are adjusted.

IV. Implementation

The development flow was as follows: create the highway with NetEdit, obtain the parameters of the simulation, create a custom Gym environment and train the networks.

1. Languages and tools:

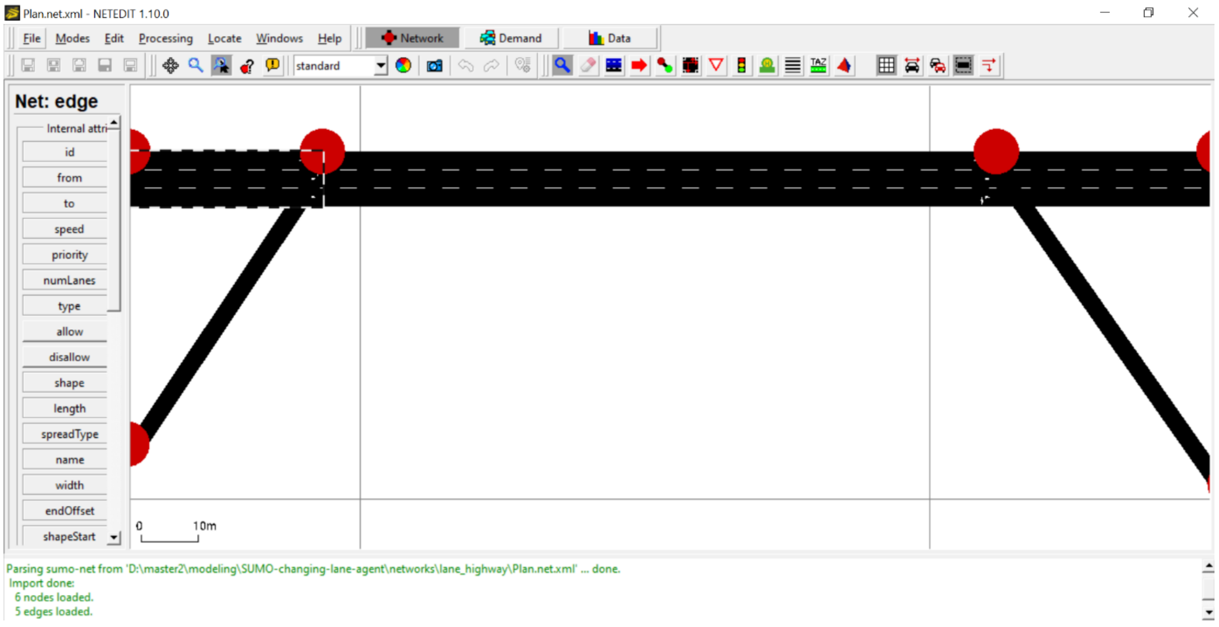
In this project we used the following tools:

python™ SUMO PC TraCi Python Gym

2. Creation of the environment

Highway building using NetEdit

To create a simple scenario, we develop a simple three-lane highway. It is easy to create a highway with NetEdit: you just have to put some points in the workspace that will create portions of streets, it is easy to control street parameters, such as maximum speed, length and number of lanes.



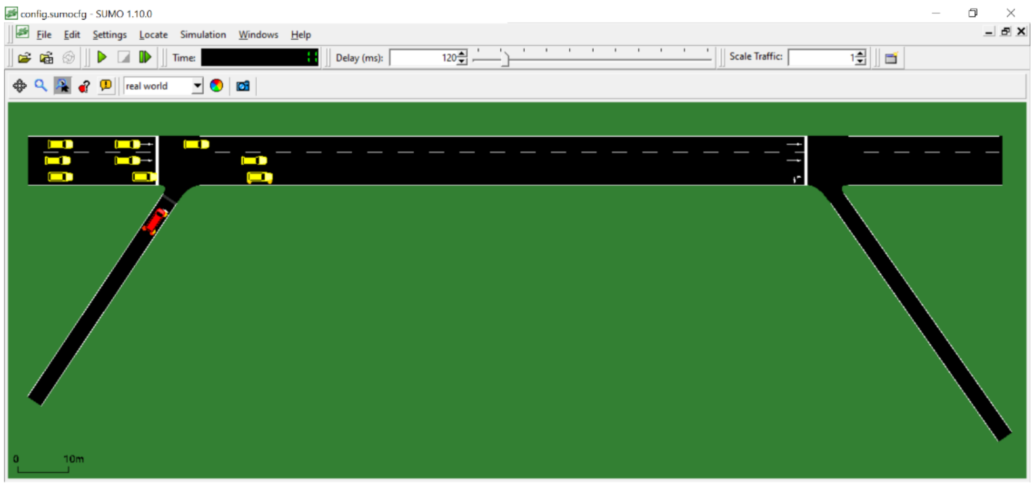
```
Parsing sumo-net from 'D:\master2\modeling\SUMO-changing-lane-agent\networks\lane_highway\Plan.net.xml' ... done.
 Import done:
   6 nodes loaded.
   5 edges loaded.
```

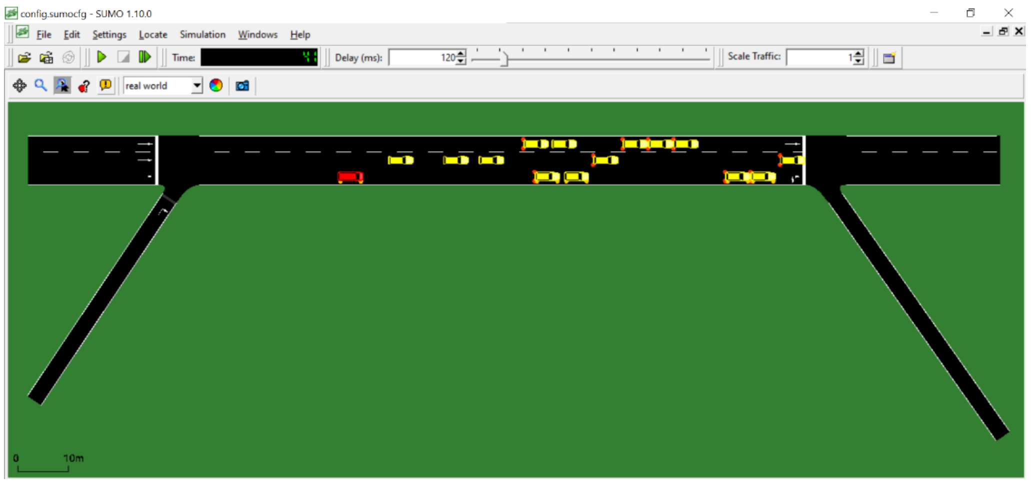
Simulation using TraCi

- Generate the xml file with config file from NetEdit.
- Run simulation from SUMO using the config file.
- SUMO's creator releases a python library to get the parameters of the simulation, namely TraCi.
- Connect the python script using TraCi to SUMO in a TCP connection.
- Create a custom Gym environment to provide state space, reward and observation based on the info that has been collected through TraCi.

Start Simulation

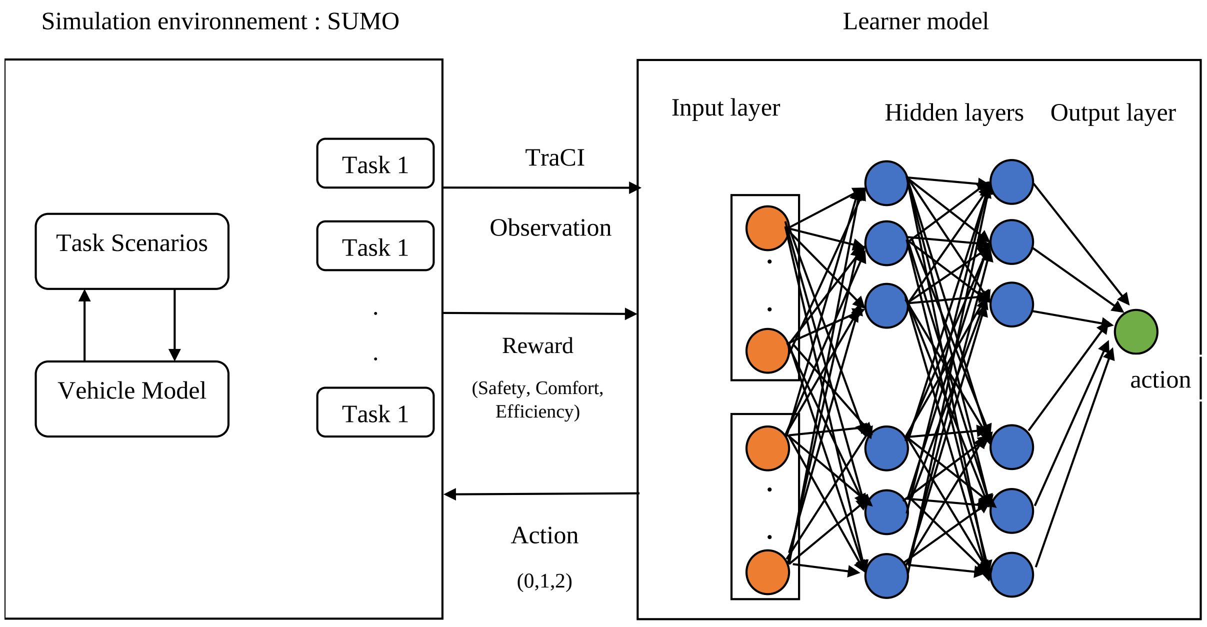
The simulation aims at making the agent run in a dense traffic scenario without collision with other vehicles. To make a realistic scenario of traffic, 35 autonomous vehicles were added to the simulation other than the target vehicle red to have a moderate traffic.


config.sumocfg - SUMO 1.10.0
File Edit Settings Locate Simulation Windows Help
Time:
Delay (ms): 120
Scale Traffic: 1
real world
0 10m


config.sumocfg - SUMO 1.10.0
File Edit Settings Locate Simulation Windows Help
Time:
Delay (ms): 120
Scale Traffic: 1
real world
0 10m

3. Global Architecture

I took inspiration from [6] , this study proposes a good strategy of rewards of lane change strategy.



There are two major components in the system: a learner model and a simulation environment. Specifically, the learner model train the target vehicle (agent) to learn while interacting with the surrounding traffic. The simulation environment, which includes the road network, traffic, and different task scenarios, is generated using a high-fidelity microscopic traffic simulation suite SUMO.

To enable safe, smooth, and efficient driving behaviors on highways, the target vehicle first receives its current state and its surrounding vehicles' state from the SUMO environment through TraCI, and these states are passed through the policy. Next, the target vehicle determines the action based on the developed policy, which then sends the action back to SUMO to model the vehicle's movement in the next time step and compute the corresponding reward.

State Space

As state space representation [6] . Starting from the position of the target vehicle, by using Traci, first we collect all the vehicles in its neighborhood (the closest in a certain radius). For each vehicle, we represented it with the following information: lateral position, speed, acceleration and distance. The target vehicle is represented with the same features but augmented with the longitudinal position. According to this, the state space is represented with a vector of 37 features: 5 features for the agent, 4 for each of the possible 8 vehicles around the ego vehicle.

Action Space

The action space is made of three choices: stay in the current lane {0}, change to the left {1} and change to the right {2}.

Reward Function

According to [6] we calculate the reward function respectively the three objectives, safety, efficiency, and comfort.

- Comfort: evaluation of jerk (lateral and longitudinal direction):

$$R_{comf}(t) = -\alpha \cdot \dot{a}_x(t)^2 - \beta \cdot \dot{a}_y(t)^2$$

  where a_x and a_y are the lateral jerk and the longitudinal jerk; and α and β are the corresponding weights for lateral and longitudinal comfort. This reward function is introduced to avoid sudden acceleration or deceleration of the vehicle that may cause vehicle occupant discomfort.
- Efficiency: evaluation of travel time and the speed:
  the efficiency reward function can be defined as:

$$\begin{cases} R_{time}(t) = -\delta t \\ R_{lane}(t) = -|P_x - P_x^*| \\ R_{speed}(t) = -|V_y - V_{desired}| \end{cases}$$

$$R_{eff}(t) = w_t \cdot R_{time}(t) + w_l \cdot R_{lane}(t) + w_s \cdot R_{speed}(t)$$

  where R_{time}(t) is the sub-reward related to time, R_{lane}(t) represents the difference between the target vehicle's lateral position with desired vehicle, while R_{speed}(t) represents the difference between the target vehicle speed with respect to the desired vehicle speed.

- Safety: evaluation of the risk of collisions and near collisions:

$$R_{collision} = \begin{cases} R_{near\_collision} & \text{if } D < d_s \\ -100 & \text{if collision} \end{cases}$$

4. DQN with replay memory Algorithm

There are two main phases that are interleaved in the Deep Q-Learning Algorithm. One is where we sample the environment by performing actions and store away the observed experienced tuples in a replay memory. The other is where we select the small batch of tuples from this memory, randomly, and learn from that batch using a gradient descent (SGD) update step.

```
Initialize network $Q$
Initialize target network $\hat{Q}$
Initialize experience replay memory $D$
Initialize the $Agent$ to interact with the Environment
while $not\ converged$ do
    /* Sample phase
    $\epsilon \leftarrow$ setting new epsilon with $\epsilon$-decay
    Choose an action $a$ from state $s$ using policy $\epsilon$-greedy$(Q)$
    $Agent$ takes action $a$, observe reward $r$, and next state $s'$
    Store transition $(s, a, r, s', done)$ in the experience replay memory $D$

    if $enough\ experiences\ in\ D$ then
        /* Learn phase
        Sample a random $minibatch$ of $N$ transitions from $D$
        for $every\ transition\ (s_i, a_i, r_i, s'_i, done_i)\ \ in\ \ minibatch$ do
            if $done_i$ then
                $y_i = r_i$
            else
                $y_i = r_i + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(s'_i, a')$
            end
        end
        Calculate the loss $\mathcal{L}$ =1/N $\sum_{i=0}^{N-1} (Q(s_i, a_i) - y_i)^2$
        Update $Q$ using the SGD algorithm by minimizing the loss $\mathcal{L}$
        Every $C$ steps, copy weights from $Q$ to $\hat{Q}$
    end
end
```

Additionally, we reinforce the algorithm with the optimization features to maximize learning speed and stability the Huber loss and Adam optimizer.

5. Results & evaluation

The highway segment length to the ramp exit is 1400 m and each lane width is 3.2 m. Vehicle counts are generated from a probability model to simulate the dense traffic, a baseline run with parameters number of vehicles is 35, Epoch= 2500, Step in epoch= 218, Replay memory start size: 33, layer1=64, layer2=128, layer3=64.

Scenario1:

Max speed target vehicle= 100, Max speed other vehicle= 120, Acceleration target= 2.5, Acceleration others= 2.5, Deceleration= 4.5, Num of vehicle= 35, Learning rate=0.0001, gamma=0.99, batch size=64, threshold distance to considered vehicle= 10, weight comfort = 0.005, weight lane = 2.5, weight speed = 2.5, weight time = 2.0, weight efficiency = 0.0005.

For scenario 2, we decreased the speed of target vehicle.

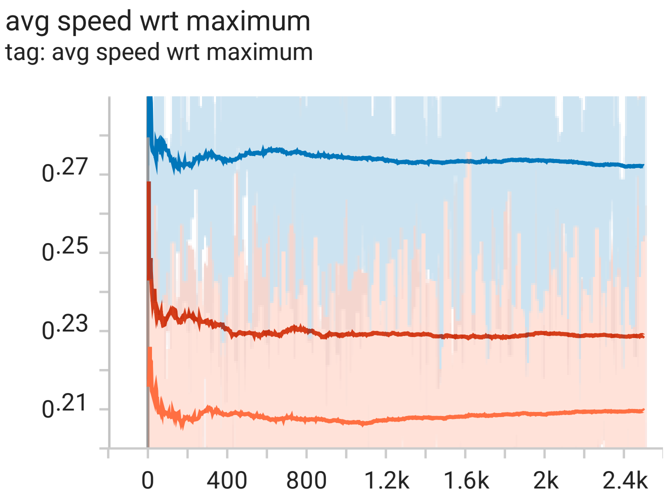
Scenario2:

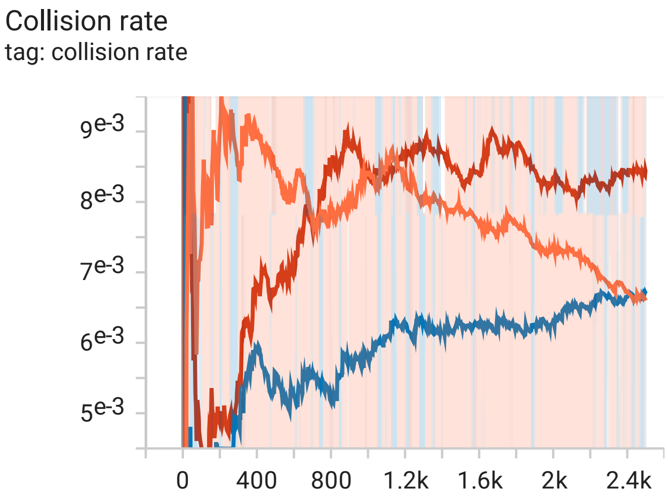
Max speed target vehicle= 50, Max speed other vehicle= 70, Acceleration target= 1.5, Acceleration others= 1.5, Deceleration= 4.5, Num of vehicle= 35, Learning rate=0.01, gamma=0.95, batch size=32, threshold distance to considered vehicle= 10, weight comfort = 0.005, weight lane = 2.5, weight speed = 2.5, weight time = 2.0, weight efficiency = 0.0005.

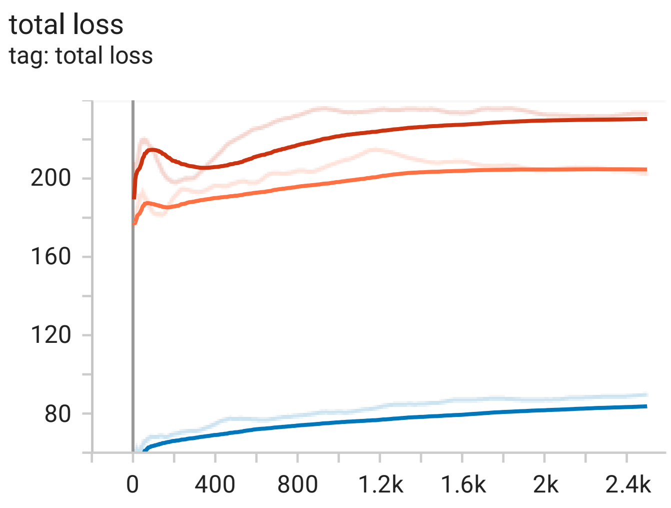
For the third scenario, we decreased weights in efficency and comfort reward so to weight more the safety.

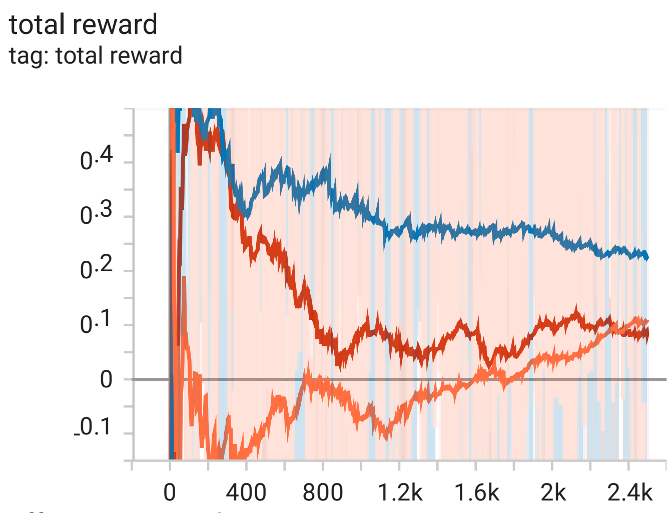
Scenario3:

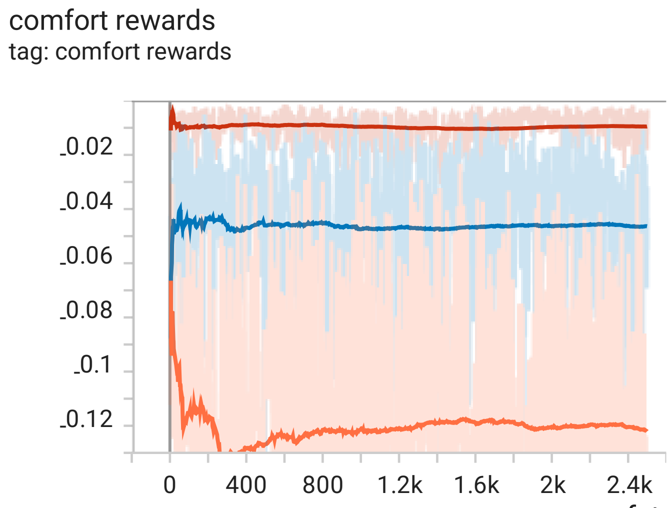
Max speed target vehicle= 90, Max speed other vehicle= 110, Acceleration target= 2.5, Acceleration others= 2.5, Deceleration= 4.5, Num of vehicle= 35, Learning rate=0.0001, gamma=0.99, batch size=64, threshold distance to considered vehicle= 10, weight comfort = 0.0005, weight lane = 1.5, weight speed = 1.5, weight time = 1.0, weight efficiency = 0.0005.

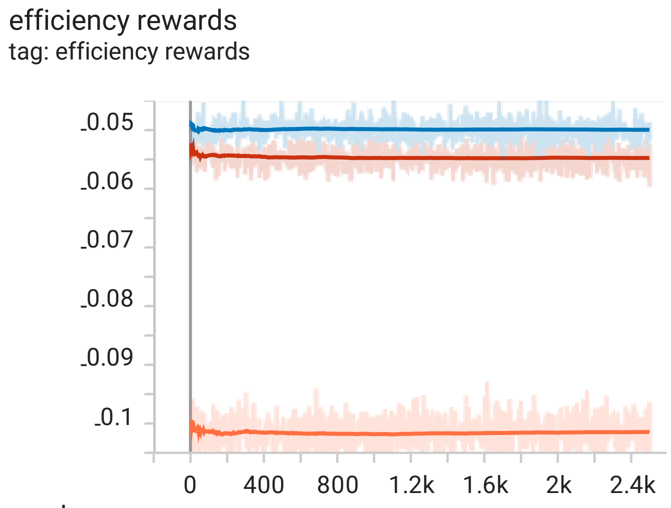


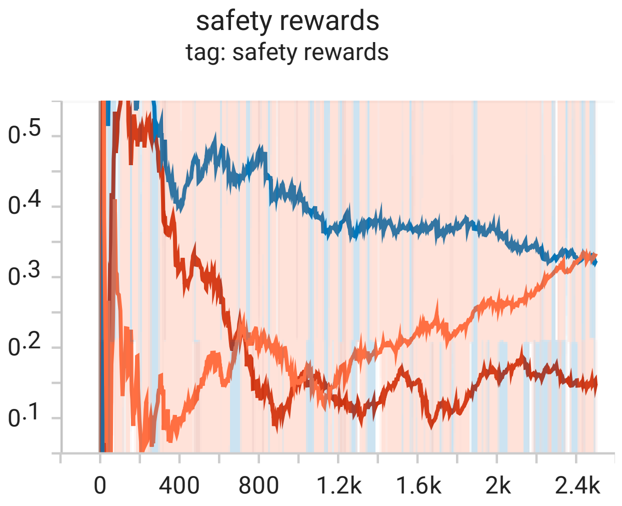












The orange line is the first scenario, the blue one is the second and the red is the third one. As you can see the best one regardless the speed, collision rate, total loss and the total rewards is the scenario 2 (blue one).

V. Conclusion

This project was about the creation of autonomous driving trained by a deep reinforcement learning algorithm. We first learned about how to create an environment using Netedit, then we read about sumo for the simulation of that network.

Next, we have read up on reinforcement learning, neural network and finally deep reinforcement learning. We apply the DQN algorithm on the network by defining actions, state space and rewards. and finally, we got the results and analyzed them.

VI. References

[1] Chen et al. “Autonomous Driving using Safe Reinforcement Learning by Incorporating a RegretbasedHuman Lane-Changing Decision Model”. In: (2019).
[2] Mirchevska et al. “High-level Decision Making for Safe and Reasonable Autonomous Lane Changing using Reinforcement Learning”. In: (2019).
[3] B Ravi Kiran et al. “Deep Reinforcement Learning for Autonomous Driving: A Survey”. In: (2020).
[4] Huegle et al. “Dynamic Input for Deep Reinforcement Learning in Autonomous Driving”. In: (2019).
[5] Huegle et al. “Dynamic Interaction-Aware Scene Understanding for Reinforcement Learning in Autonomous Driving”. In: (2020).
[6] Ye et al. “Automated Lane Change Strategy using Proximal Policy Optimization-based Deep Reinforcement Learning”. In: (2019).